Interprocedural Specialization of Higher-Order Dynamic Languages Without Static Analysis (Artifact)

Baptiste Saleil[1] **and Marc Feeley**[2]

1 **Université de Montréal**
Montreal, Quebec, Canada
baptiste.saleil@umontreal.ca
2 **Université de Montréal**
Montreal, Quebec, Canada
feeley@iro.umontreal.ca

Abstract

This artifact is based on LC, a research oriented JIT compiler for Scheme. The compiler is extended to allow interprocedural, type based, code specialization using the technique and its implementation presented in the paper. Because the technique is directly implemented in LC, the package contains the build of the compiler used for our experiments. To support repeatability, the artifact allows the user to easily extract the data presented in the paper such as the number of executed type checks or the generated code size. The user can repeat the experiments using a set of standard benchmarks as well as its own programs. Instructions for building the compiler from scratch are also provided.

1998 ACM Subject Classification D.3.4 Processors

Keywords and phrases just-in-time compilation, interprocedural optimization, dynamic language, higher-order function, scheme

Digital Object Identifier 10.4230/DARTS.3.2.14

Related Article Baptiste Saleil and Marc Feeley, "Interprocedural Specialization of Higher-Order Dynamic Languages Without Static Analysis", in Proceedings of the 31st European Conference on Object-Oriented Programming (ECOOP 2017), LIPIcs, Vol. 74, pp. 23:1–23:23, 2017.
http://dx.doi.org/10.4230/LIPIcs.ECOOP.2017.23

Related Conference European Conference on Object-Oriented Programming (ECOOP 2017), June 18-23, 2017, Barcelona, Spain

1 Scope

The artifact is designed to support repeatability of all the experiments presented in the paper. Users can obtain the result from the metrics used for the experiments such as the number of executed type checks, the size of the generated machine code or the execution time using a set of standard benchmarks as well as their own Scheme programs. A tool allowing to automatically gather the data presented in the paper is also included in the artifact.

2 Content

This artifact contains a build of the LC compiler used for the experiments presented in the paper. The archive contains a virtual appliance packaged using the Open Virtualization Format (OVF) and the artifact documentation. This appliance can be imported and started using the virtualization software *VirtualBox*.

When the system is booted, the user can find the artifact in a folder named `artifact` on the desktop. To facilitate the use of the artifact, two windows are opened on boot:

- A file explorer window showing the content of the `artifact` folder
- A terminal window opened in the `artifact` folder

The `artifact` folder contains:

- The artifact documentation
- The paper
- The sources of the compiler
- The benchmarks used for the experiments

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). The latest version of the compiler is available on Github: `https://github.com/bsaleil/lc`

4 Tested platforms

Because the compiler targets x86-64 assembly, the artifact must be executed on a platform using a x86-64 CPU supporting SSE extensions. The artifact is known to work on any operating system satisfying this condition and running Oracle VirtualBox version 5 (`https://www.virtualbox.org/`) with at least 8 GB of free space on disk and at least 3 GB of free space in RAM.

5 License

BSD-3-Clause (`https://opensource.org/licenses/BSD-3-Clause`)

6 MD5 sum of the artifact

7a431b74c95241dc09445eeed4790ca0

7 Size of the artifact

2.8 GB